Aventura 3

Hilos y exclusión mutua

En esta aventura lanzaremos 10 hilos que accederán concurrentemente N veces a una pila (la implementada en la aventura 1), y en cada ocasión:

- extraerán el contenido del puntero que esté en la cima (nº entero),
- lo incrementarán en una unidad
- y lo volverán a dejar en la pila.

¿Cuál es el reto de esta aventura?

Tenemos un recurso compartido (pila) que hay que proteger del acceso concurrente de los diversos hilos.

¿Qué conceptos trabajamos con estos retos?

- Procesos e Hilos en C de Unix/Linux y Semáforos
- Librería pthread.h. Funciones **pthread_create**(), **pthread_mutex_lock**(), **pthread_mutex_unlock**(), **pthread_exit**(), **pthread_join**(), **pthread_self**(). Macro **PTHREAD_MUTEX_INITIALIZER**

¿Cuáles son los detalles de implementación?

PASOS:

1. **Preparar la pila (tendrá que contener tantos elementos como hilos consideremos).**
   Al comenzar, y antes de crear los 10 hilos, el hilo principal de **av3.c** deberá verificar si la pila, cuyo nombre del fichero se ha pasado por consola (argv[1]), existe, en caso contrario deberá crearla (inicializada con punteros a 0's)[1]. Si

[1] Recordad que los datos que se pasen a my_stack_push() se han de haber declarado tipo puntero, por ejemplo en este caso: int *data;. Para inicializar a 0 el contenido de cada puntero que meteremos en la pila (*data=0;), hay que reservar previamente espacio para cada entero con malloc(). Como buenas prácticas de programación, posteriormente habría que liberar todo el espacio reservado con malloc() (incluído también el que han reservado funciones de my_lib.c), y eso se puede hacer

ya existe hay que usar esa, sin machacarla. Si la pila tiene menos de 10 elementos, se deberán agregar los restantes individualmente, con un único valor entero de cero. Si la pila contiene más de 10 elementos, se ignoran el resto.

Para gestionar la pila disponéis de las siguientes funciones ya implementadas en la Aventura 1:

- **my_stack_init()**
- **my_stack_len()**
- **my_stack_push()**
- **my_stack_pop()**
- **my_stack_purge()**
- **my_stack_read()**
- **my_stack_write()**

Si no se ha especificado un nombre de fichero como argumento del programa[2], se ha de mostrar un mensaje de error **notificando la sintaxis correcta**

2. **Crear los hilos.**
   El hilo principal ha de crear los 10 hilos (NUM_THREADS = 10).

   Para ello tenéis que utilizar un bucle llamando a la función **pthread_create**(). Tendremos un array de tipo *pthread_t* para almacenar los identificadores de los hilos. Como primer argumento de pthread_create() hay que pasarle el puntero al elemento correspondiente de ese array.
   Como tercer argumento hay que indicarle el nombre de la función que ejecutará el hilo (sin argumentos ni paréntesis).
   El 2ª y 4ª argumento pueden ser NULL.

3. **Implementar la función que han de ejecutar los hilos**
   Se habrá definido un semáforo global de tipo *pthread_mutex_t,* el cual se inicializará con la macro **PTHREAD_MUTEX_INITIALIZER** de la librería **pthread.h**.

   Se creará una función tipo puntero a cualquier tipo de datos (que es la que se indicará como argumento en pthread_create()).

```
void *worker(void *ptr)
```

fácilmente llamando a **my_stack_purge()**, cuando ya se haya volcado la pila en el fichero. (Ver discusión)

[2] int main (int argc, char *argv[])

Se implementará un bucle con N iteraciones (donde N podría ser = 1.000.000)[3]. En cada iteración se extraerá un elemento de la pila, **se incrementará en uno** el valor del entero de datos, y se volverá a insertar en la pila. Se ha de crear **una sección crítica para realizar el pop** y **otra para realizar el push.**
Se utilizarán las funciones **pthread_mutex_lock**() y **pthread_mutex_unlock**(), pasándoles la dirección de ese semáforo, para definir las 2 secciones críticas donde se insertarán las funciones **my_stack_pop()** y **my_stack_push()** respectivamente.
Para salir de la función se llamará a **pthread_exit**(NULL).

4. **Volcar la pila en un fichero.**
   Cuando estén todos los hilos creados, el hilo principal se bloqueará hasta que acaben todos los hilos, y posteriormente guardará la pila en un fichero.

   El hilo principal se bloquea hasta que acaben todos los hilos mediante la función **pthread_join**()[4].
   Se utilizará la función **my_stack_write()** para guardar la pila en el fichero.
   Para finalizar se llamará a **pthread_exit()**.

5. **Comprobar los datos del fichero.**
   Con otro programa independiente, **reader**, se leerá el fichero almacenado, y se imprimirá los valores de cada elemento de la pila actualizado por los hilos, la suma, el mínimo, el máximo y la media.[5]

   Deberíais obtener todos los cálculos en un solo bucle.
   Para obtener el mínimo podéis inicializar una variable de tipo entero con el valor **INT_MAX** (el máximo valor que puede tomar un entero) de la librería **limits.h**

Ejemplo de funcionamiento reducido (NUM_THREADS = 3, N = 5) para observar la intercalación de hilos y operaciones pop y push.
Observación: Para que podamos observar más intercalación en la ejecución de los hilos se ha introducido un sleep(0.001) entre las dos secciones críticas, tras

[3] Comprobad si de esa manera la ejecución dura aproximadamente entre 5 y 10 segundos y si no adaptarlo. Podéis obtener el tiempo de ejecución mediante el comando time: $time ./av3 pila. Básicamente se trata de que se llegue a poder vaciar la pila y no queden elementos con valor 0.
[4] Es un bucle posterior y diferente del que llama a pthread_create().
[5] Si la pila tiene más de 10 elementos, los cálculos sólo tendrán en cuenta los 10 primeros.

incrementar el valor del data (sólo para este ejemplo, no cuando N=1.000.000 que ya se producirá intercalación de forma natural!!!)

```
uib ~$ ./av3  s4
Threads: 3, Iterations: 5
stack->size: 4
original stack lenght: 0
new stack length: 3
0) Thread 140607035332352 created
1) Thread 140607026939648 created
Soy el hilo 140607026939648 ejecutando pop
2) Thread 140607018546944 created
Soy el hilo 140607035332352 ejecutando pop
Soy el hilo 140607018546944 ejecutando pop
Soy el hilo 140607026939648 ejecutando push
Soy el hilo 140607035332352 ejecutando push
Soy el hilo 140607035332352 ejecutando pop
Soy el hilo 140607026939648 ejecutando pop
Soy el hilo 140607018546944 ejecutando push
Soy el hilo 140607018546944 ejecutando pop
Soy el hilo 140607035332352 ejecutando push
Soy el hilo 140607035332352 ejecutando pop
Soy el hilo 140607026939648 ejecutando push
Soy el hilo 140607026939648 ejecutando pop
Soy el hilo 140607018546944 ejecutando push
Soy el hilo 140607018546944 ejecutando pop
Soy el hilo 140607035332352 ejecutando push
Soy el hilo 140607035332352 ejecutando pop
Soy el hilo 140607026939648 ejecutando push
Soy el hilo 140607026939648 ejecutando pop
Soy el hilo 140607035332352 ejecutando push
Soy el hilo 140607035332352 ejecutando pop
Soy el hilo 140607018546944 ejecutando push
Soy el hilo 140607018546944 ejecutando pop
Soy el hilo 140607026939648 ejecutando push
Soy el hilo 140607018546944 ejecutando push
Soy el hilo 140607018546944 ejecutando pop
Soy el hilo 140607035332352 ejecutando push
Soy el hilo 140607026939648 ejecutando pop
Soy el hilo 140607026939648 ejecutando push
Soy el hilo 140607018546944 ejecutando push
Written elements from stack to file: 3
Released bytes: 76
```

[6] El identificador del hilo en ejecución lo podemos obtener con la función **pthread_self**(). Sería equvalente a getpid() para procesos.

```
Bye from main
uib ~$ ./reader s4
Stack length: 3
5
6
4
Items: 3 Sum: 15 Min: 4 Max: 6 Average: 5
```

¿Qué recursos metemos en nuestras mochilas?

Documentos de apoyo:

- Procesos e Hilos en C de Unix/Linux
- Hilo de ejecución (wikipedia)
- Pthread Example
- Sincronización de hilos en POSIX: Mutex
- Semaphore Example
- POSIX Threads, Hilos o Hebras
- Hilos de ejecución POSIX
- http://cortesfernando.blogspot.com.es/2011/11/linux-threads.html
- POSIX Threads Programming
- Synchronizing Threads with POSIX Semaphores
- POSIX semaphores

Además podéis usar este Makefile:

```
C=gcc
CFLAGS=-c -g -Wall -std=c99
LDFLAGS=-pthread

SOURCES=my_lib.c av3.c reader.c
LIBRARIES=my_lib.o
INCLUDES=my_lib.h
PROGRAMS= av3 reader
OBJS=$(SOURCES:.c=.o)

all: $(OBJS) $(PROGRAMS)

#$(PROGRAMS): $(LIBRARIES) $(INCLUDES)
#	$(CC) $(LDFLAGS) $(LIBRARIES) $@.o -o $@


av3: av3.o $(LIBRARIES) $(INCLUDES)
	$(CC) $(LDFLAGS) $(LIBRARIES) $< -o $@
```

```
reader: reader.o $(LIBRARIES) $(INCLUDES)
  $(CC) $(LDFLAGS) $(LIBRARIES) $< -o $@

%.o: %.c $(INCLUDES)
  $(CC) $(CFLAGS) -o $@ -c $<

.PHONY: clean
clean:
  rm -rf *.o *~ $(PROGRAMS)
```

¿Cómo comprobamos que hemos superado este nivel?

Podéis ejecutar este test:

test.sh

```
make
clear

echo "./av3 ##################################################"
./av3
echo "./reader ###############################################"
./reader
echo "./reader s #############################################"
./reader s

echo "time ./av3 s0 # pila inexistente ########################"
time ./av3 s0
echo "./reader s0 ############################################"
./reader s0

echo "time ./av3 s0 # 2ª ejecución misma pila #################"
time ./av3 s0
echo "./reader s0 ############################################"
./reader s0

echo "./av3 s1-14el ##########################################"
# pila existente con 14 elementos
./av3 s1-14el
echo "./reader s1-14el #######################################"
# los calculos han de hacerse sobre los 10 superiores
./reader s1-14el

echo "./av3 s2-6el ###########################################"
#pila existente con 6 elementos
./av3 s2-6el
```

```
echo "./reader s2-6el ####################################################"
./reader s2-6el
echo "./av3 s2-6el ######################################################"
./av3 s2-6el
echo "./reader s2-6el ####################################################"
echo "los cálculos han de añadir 10.000.000 a la suma anterior"
./reader s2-6el
```

y contrastar vuestros resultados con estos:

```
uib ~$ ./test.sh
gcc -c -g -Wall -std=c99 -o av3.o -c av3.c
gcc -pthread my_lib.o av3.o -o av3

./av3 ##############################################
USAGE:          ./av3 filename
./reader ###########################################
USAGE:          ./reader filename
./reader s #########################################
Couldn't open stack file s
time ./av3 s0 # pila inexistente ###################
Threads: 10, Iterations: 1000000
stack->size: 4
original stack content:
original stack length: 0
stack content for treatment:
0
0
0
0
0
0
0
0
0
0
new stack length: 10
0) Thread 140316179953408 created
1) Thread 140316171560704 created
2) Thread 140316163168000 created
3) Thread 140316154775296 created
4) Thread 140316146382592 created
5) Thread 140316137989888 created
6) Thread 140316129597184 created
7) Thread 140316121204480 created
```

```
8) Thread 140315776251648 created
9) Thread 140315767858944 created
stack content after threads iterations:
1047674
1041481
1065748
1023392
1010130
970752
998398
923245
954990
964190
stack length: 10
Written elements from stack to file: 10
Released bytes: 216
Bye from main


real    0m4,204s
user    0m8,232s
sys     0m7,940s
./reader s0
####################################################
Stack length: 10
1047674
1041481
1065748
1023392
1010130
970752
998398
923245
954990
964190
Items: 10 Sum: 10000000 Min: 923245 Max: 1065748 Average: 1000000

time ./av3 s0 # 2ª ejecución misma pila ############################
Threads: 10, Iterations: 1000000
original stack content:
1047674
1041481
1065748
1023392
1010130
```

```
970752
998398
923245
954990
964190
original stack length: 10
0) Thread 140087132628736 created
1) Thread 140087124236032 created
2) Thread 140087115843328 created
3) Thread 140087107450624 created
4) Thread 140087099057920 created
5) Thread 140087090665216 created
6) Thread 140087082272512 created
7) Thread 140086867916544 created
8) Thread 140086859523840 created
9) Thread 140086851131136 created
stack content after threads iterations:
2145881
2033516
2024085
2095640
2079980
1975054
1951380
1947374
1876366
1870724
stack length: 10
Written elements from stack to file: 10
Released bytes: 216
Bye from main

real    0m4,619s
user    0m9,049s
sys     0m8,637s
./reader s0
#####################################################
Stack length: 10
2145881
2033516
2024085
2095640
2079980
1975054
1951380
```

```
1947374
1876366
1870724
Items: 10 Sum: 20000000 Min: 1870724 Max: 2145881 Average: 2000000

./av3 s1-14el0 ################################################
Threads: 10, Iterations: 1000000
original stack content:
0
0
0
0
0
0
0
0
0
0
0
0
0
0
original stack length: 14
0) Thread 140431507162880 created
1) Thread 140431498770176 created
2) Thread 140431490377472 created
3) Thread 140431481984768 created
4) Thread 140431270606592 created
5) Thread 140431127996160 created
6) Thread 140431262213888 created
7) Thread 140431253821184 created
8) Thread 140431245428480 created
9) Thread 140431237035776 created
stack content after threads iterations:
1124121
1047052
991810
1020130
992934
979492
970612
943215
962888
967746
0
```

```
0
0
0
stack length: 14
Written elements from stack to file: 14
Released bytes: 296
Bye from main

./reader s1-14el0
################################################
Stack length: 14
1124121
1047052
991810
1020130
992934
979492
970612
943215
962888
967746
Items: 10 Sum: 10000000 Min: 943215 Max: 1124121 Average: 1000000

./av3 s2-6el0
#####################################################
Threads: 10, Iterations: 1000000
original stack content:
0
0
0
0
0
0
original stack length: 6
stack content for treatment:
0
0
0
0
0
0
0
0
0
0
```

```
stack length: 10
0) Thread 139990235432704 created
1) Thread 139990227040000 created
2) Thread 139990218647296 created
3) Thread 139990210254592 created
4) Thread 139990201861888 created
5) Thread 139990193469184 created
6) Thread 139990185076480 created
7) Thread 139990176683776 created
8) Thread 139989962716928 created
9) Thread 139989954324224 created
stack content after threads iterations:
1072638
1082805
1100541
991967
1007272
998794
977967
970092
926949
870975
stack length: 10
Written elements from stack to file: 10
Released bytes: 216
Bye from main

./reader s2-6el0
##################################################
Stack length: 10
1072638
1082805
1100541
991967
1007272
998794
977967
970092
926949
870975
Items: 10 Sum: 10000000 Min: 870975 Max: 1100541 Average: 1000000
```

Observaciones

- Si la pila estaba vacía antes de ejecutar av3, la suma ha de ser igual al número de iteraciones * número de hilos, en el ejemplo 1.000.000 * 10 y la media ha de ser igual al número de iteraciones, en el ejemplo 1.000.000.
- Si volvemos a lanzar av3, incrementando los valores de la pila anterior, al ejecutar el reader, la suma y la media darían el doble, y así sucesivamente…
- El reader no ha de abortar con pila inexistente ni con pila con 0 o menos de 10 elementos[7].

[7] Podéis generar una con vuestro mismo programa poniendo un nº de hilos inferior a 10